Skin Cancer Surface Shape Based Classification
User guide

Steven McDonagh (0458953)

March 14, 2008

Contents

1 Introduction **3**

2 Installation **3**

3 Preparing new data **4**
3.1 File path issues 4
3.2 Using the image masking tool 4

4 Adding a new feature **6**
4.1 Extracting all currently defined features 6
4.2 Loading the feature data and adding a single new feature 7
4.3 trainingdata.mat 7

5 Feature selection **8**
5.1 Greedy selection 8
5.2 Exhaustive selection 8

6 Training the system and classification **9**
6.1 Classification commands 9
6.2 Classification evaluation 9

7 Miscellaneous useful bits **10**

8 Summary **10**

1 Introduction

This user guide provides a brief overview of the system implemented as part of *Skin Cancer Surface Shape Based Classification* [1, 2], an undergraduate project undertaken within the School of Informatics. This document contains hands-on instructions for carrying out various tasks with the system, including preparing new data for the purpose of training and classification, adding additional features (measurable properties of the data), performing feature selection, training the implemented classifier and performing classification experiments. Matlab commands and examples are given in `teletype` where appropriate. The system was primarily implemented and made use of with Matlab 7.4.0.336 (R2007a) running on Dice Linux. The guide is written with this environment in mind.

Please feel free to contact the author at **s0458953@sms.ed.ac.uk** with any queries, problems, comments or suggestions for future versions of this document.

2 Installation

The system is written in Matlab and therefore the installation process simply involves adding the relevant source directories to the Matlab path. An easy way to accomplish this is to right click the top level source directory within Matlab and select *Add to path* → *Selected Folders and Subfolders*. The contents of the top level source directory for the project (correct as of 13/03/08) can be seen in Figure 1. For the remainder of this document we assume the top level source directory is named "**../src/**".

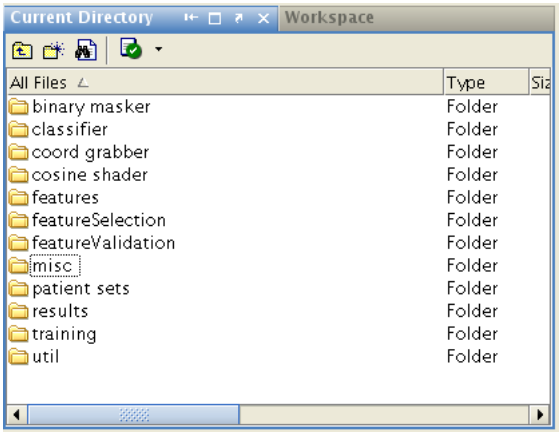

Figure 1: Top level source tree

3 Preparing new data

Preparing new data for the system essentially involves creating segmentation masks for the image data. For this purpose a simple image masking tool was created. This tool can be invoked by calling the function `binarymasker()` which by default is located in **../src/binary masker/binarymasker.m**.

3.1 File path issues

The masking tool was written with the expectation that multiple images will be masked in a single session. To this end the tool attempts to mask all images found within the subdirectories of a specified base file path. This base file path is currently:

/group/project/VISION/web/3D_SKIN_DATA/BATCH2/

and will likely need to change depending on the location of source images.

Image loading is performed by regular expression matching and is dependent on the current patient file naming conventions. If file naming conventions change, it is likely that the image loading code within the binarymasker.m file will need to be adapted accordingly.

Successfully created image masks are automatically written to disk as PNG files. The base directory these files are written to is currently:

/group/project/VISION/web/MCDONAGH/BATCH2_masks/

Again, this path should be modified to accommodate read/write permissions if need be.

3.2 Using the image masking tool

Once the masking tool has been invoked and an image successfully loaded to mask, the tool should appear similar to Figure 2. The image to be masked is displayed in two windows. The **upper** image window is where masking and zooming is performed with the **lower** display providing an unzoomed overview of the image. The user is then able to introduce draggable control points on the **upper** of the two windows for the purpose of image masking. Once the area to be masked has been indicated by a set of control points the "*Done*" button (green tick icon) located in the task bar confirms the set of points. The control points are confirmed in **three stages** corresponding to the **three masks** generated for *spot area*, *uncertain area* and *normal skin area* masks in turn with the "*Done*" button being used three times to confirm the three sets of points corresponding to each mask in turn. The tool expects the masks to be allocated in this order (*1. spot, 2. uncertain, 3. normal skin)*. The polygons defined by connecting the sets of confirmed control points are then used to generate the masks for the image. After the three masks have been defined

a final confirmation and minidisplay of the selected masks will be displayed. Confirming the masks will write three PNG masks to disk and declining will reset the process.

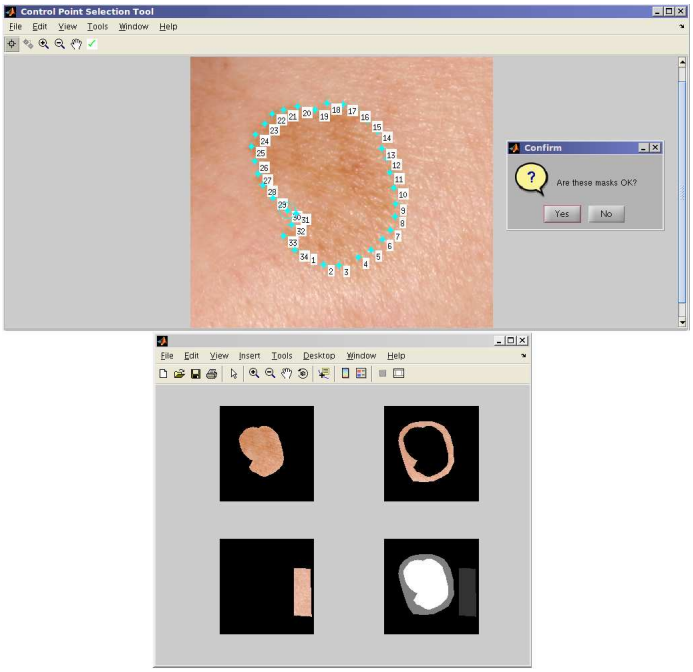

Figure 2: Binary masking tool

Step by step this process can be summarised as follows:

1. **Spot mask:** Define a set of control points surrounding the spot.
2. Click *"Done"*
3. **Uncert mask:** Define a second set of control points **encompassing** the first by a small margin. (This mask is used to highlight an in between uncertain region).
4. Click *"Done"*
5. **Normal skin mask:** Define a set of control points covering only normal skin in the image. This area should not overlap with the previous two masks.
6. Click *"Done"*
7. View the created masks in the mini preview and confirm if satisfied.

Once the masks have been confirmed, the image tool finds the next available patient image sample and repeats the process. The tool will automatically quit once all available samples below the specified base file path directory have been exhausted. Sample resultant image masks are shown in Figure 3.

Figure 3: Sample binary masks

4 Adding a new feature

4.1 Extracting all currently defined features

New features can be added by modifying the file **../src/features/properties.m**. This method passes the relevant patient data to individual feature extraction functions and collects the feature values which are then added to a feature vector variable (found at the bottom of this file) named `featureVec`.

The data from which features can be extracted is held in the variables `skinIm,spotIm,uncertIm` (matrices representing the intensity masks) and `xData`, `yData`, `zData` (matrices containing the depth data for the sample).

It is recommended that the majority of feature extraction work be written in a separate function which is passed the above data variables as arguments. For example a new feature computing some measure of the *z-depth* values within the spot mask area might be written in a function `newFeature` and then calculated and added to the feature vector in **properties.m** as follows:

```
newFeatureValue = newFeature(spotIm,zData);

featureVec = [feature1,feature2,feature3,...,newFeatureValue];
```

Once the **properties.m** file has been updated, all currently defined features are extracted from the data set by calling the function:

`extractFeatures(patientFilePath)`.

The argument to the function, `patientFilePath` is a string containing the path

to a file which lists the entries of the data set. The current path that is used is given below. This variable will need to be defined. The file pointed to contains 234 patient filenames and the corresponding skin lesion classes. This file should be updated accordingly if new data samples are used.

```
patientFilePath = '../src/patient sets/234-PatientSet-SCC-SK-ML-BCC-AK';
```

Once the extraction process is complete, the extracted features of each sample and corresponding classes are written to the file ../src/training/trainingdata.mat

WARNING: The current feature set incorporates some computationally expensive feature calculations that iterate over each pixel in each image in the data set (e.g. the *texture ratio* features). Extracting the full set of features is therefore likely to take several hours (assuming an Intel(R) Dual Core CPU @ 1.86GHz running Dice Linux or equivalent).

4.2 Loading the feature data and adding a single new feature

Due to the noted computational expense of extracting all features from the data set, a single feature can be defined, extracted from the data set and the feature set updated using the function `featureAdder`. This involves editing the **../src/features/featureAdder.m** function lines found below:

```
% REPLACE RIGHT HAND SIDE WITH NEW FEATURE FUNCTION
newFeature(i) = abs3DMoments(spotIm,xData,yData,zData);
```

In this file the `abs3Dmoments` feature extraction function should be replaced with the name of the function which extracts the newly added feature. Calling **featureAdder.m** as below will then extract this new property from all images in the data set and update the feature set file trainingdata.mat appropriately.

```
featureAdder(patientFilePath)
```

The `patientFilePath` argument is defined in Section 4.1.

4.3 trainingdata.mat

The file ../src/training/trainingdata.mat essentially contains all the extracted information from the data set. A backup copy of this file is found in the same directory named trainingdata.backup - in case things go wrong. The variables in this Matlab data file are briefly explained in Table 1:

Variable	Current size / Value	Description
classVec	234×1 double	Integer list corresponding to sample class
featureVecs	234×30 double	Extracted feature values
numFeatures	30	Current number of extracted features
patients	234×1 cell	Patient file names
trainClassSet	5×1 cell	Classes in the training set

Table 1: trainingdata.mat variable description

5 Feature selection

5.1 Greedy selection

A greedy algorithm is available to perform best feature subset selection. Note that this algorithm does not explore the entire feature subset space and may not find the globally optimal subset combination. See [2] for further discussion of this point. Running greedy feature selection is fairly simple and just involves invoking the file **../src/featureSelection/greedySelection.m** in the manner described below.

```
greedySelection(featureSet)
```

The `featureSet` argument is a vector of indices constraining the pool of features that the algorithm is able to select from. For example `greedySelection([1:30])` will allow the algorithm to pick any of the features $1-30$ (presuming 30 features are available). Some useful feature ranges for the original trainingdata.mat file are provided below:

```
featureSet = [1:30]; % All features
featureSet = [2:13,15:17,19:21,23:25]; % features2d
featureSet = [1,14,18,22,26:30]; % features3d
```

Parameters within the **greedySelection.m** file which might be experimented with include `accuracy` which is a boolean flag dictating which criterion function to use during the search (1 = accuracy metric , 0 = misclassification cost) and `MAXSUBSETSIZE` which dictates how large the returned subset should be. This function returns the best subset found as a vector of feature indices. Again it should be noted that since each subset takes > 15 seconds to evaluate (due to the leave one out classification method used) finding an optimal subset of a reasonable size (e.g. 10 features) is a matter of hours on a standard Dice Linux machine.

5.2 Exhaustive selection

A further search algorithm for exhaustive search of the feature subset space is found in **../src/featureSelection/exhaustiveSelection.m**. Due to the

computational complexity of an exhaustive search, some initial exploration into running this function across multiple Matlab instances was made but using this method is currently not very feasible without further development. The algorithm attempts to record the search results found and has some basic capability for distributing the load across multiple instances provided by the function arguments. Check **exhaustiveSelection.m** for further info.

6 Training the system and classification

Due to the leave-one-out k-fold cross validation method of classification, the system training and classification processes are fairly intertwined. The cross validation method employed means that each classification system is trained on all of the available skin lesions *apart from the one that is to be classified*. This process is carried out by using the **../src/classifier/kfold.m** function. The `kfold` function takes a feature set and a boolean flag indicating which classifier decision rule to make use of (1 = accuracy metric, 0 = cost function metric). The function returns a confusion matrix of classification results and a cell of the classes used for the classification experiment. File paths for both the patient list and extracted feature data may need to be set within this file. Some examples of how this function might be called can be seen below.

6.1 Classification commands

The first call to `kfold` below would perform classification experiments using features $\{10, 9, 8, 3, 22, 30, 21, 25, 15, 26\}$ using the standard accuracy based decision rule. The results are found in the variable `confusionMatrix`.

```
[confusionMatrix,trainClassSet] = kfold([10,9,8,3,22,30,21,25,15,26],1);
```

The second call to `kfold` below would perform classification experiments using features $\{7, 6, 19, 22, 4, 24, 8, 1, 26, 2\}$ using the loss function based decision rule. The results are again found in the variable `confusionMatrix`.

```
[confusionMatrix,trainClassSet] = kfold([7,6,19,22,4,24,8,1,26,2],0);
```

6.2 Classification evaluation

Suggestions for how to quickly evaluate the classification results are as follows:

- `trace(confusionMatrix) / sum(sum(confusionMatrix))` - a standard $\frac{correct}{total}$ accuracy.
- `sum(sum(confusionMatrix .* lossMatrix(-1,-1)))` - misclassification cost. Here `lossMatrix(-1,-1)` is a function that returns the misclassification cost matrix when called with these arguments.
- `accuracyMetric(confusionMatrix)` - returns the weighted accuracy described in [2]

7 Miscellaneous useful bits

In this section follows a description of a few of the more useful utility functions that were written during the course of the project.

- `showPatient('P175')` - Will show the depth data (pre and post global orientation) for the patient name passed in the string argument (patient P175 in this case).
- `featureStats(patientFilePath,[1,3,5,7])` - Will plot 1D distributions of the feature values stored in trainingdata.mat. `patientFilePath` is defined as in Section 4.1. The second argument specifies which image features to plot, in the example given features $\{1, 3, 5, 7\}$ would be plotted.

.

8 Summary

This guide has provided a brief overview to the main functionality provided by the classification system developed as part of the related project. If any of the instructions or examples provided here are unclear please do not hesitate to contact the author (**s0458953@sms.ed.ac.uk**) for further advice or assistance.

References

[1] http://homepages.inf.ed.ac.uk/mcryan/projs0708/project.php?number=P090.

[2] S. McDonagh. Skin Cancer Surface Shape Based Classification. *Undergraduate thesis, School of Informatics*, 2008.